Natural

Natural Optimizer Compiler

Version 9.1.2

October 2023

ADABAS & NATURAL

This document applies to Natural Version 9.1.2 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: NATMF-NOC-912-20241106

Table of Contents

Preface

This documentation for Natural Optimizer Compiler describes various aspects which should be taken into consideration when the Natural Optimizer Compiler is installed at your site.

In the remainder of the Natural Optimizer Compiler documentation the Natural Optimizer Compiler is also referred to as NOC, which is the product code.

For an explanation of the format abbreviations used in this documents, see the section *Possible Formats* in the Natural *Statements* documentation.

General Information	Various aspects of the Natural Optimizer Compiler and how to benefit most from the Natural Optimizer Compiler.
Using the Optimizer Compiler	Statements and programs used for compilation. Statistical data on programs suitable for processing by the Natural Optimizer Compiler: `NOCSTAT` command. Examples of when to use the Optimizer Compiler.
Activating the Optimizer Compiler	How to switch on the Natural Optimizer Compiler.
Optimizer Options	Various options of the Natural Optimizer Compiler. How to apply `PGEN` to output generated code and internal Natural structures for examination. Influence by other Natural parameters.
Performance Considerations	How to achieve best performance considering data formats, arrays, alpha fields, `DECIDE ON` and numeric values.
Listing Zaps	How to receive an overview of the Zaps that have been applied to the Natural Optimizer Compiler.
Natural Optimizer Compiler Version 8.3/8.4 - Documentation Updates	Documentation updates that only apply to Natural Optimizer Compiler Version 8.3 and Version 8.4.

Related Documentation:

Installing the Natural Optimizer Compiler on z/OS, z/VSE and BS2000 in the Natural *Installation* documentation

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://techcommunity.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://containers.softwareag.com/products** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software GmbH products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

I NOC - General Information

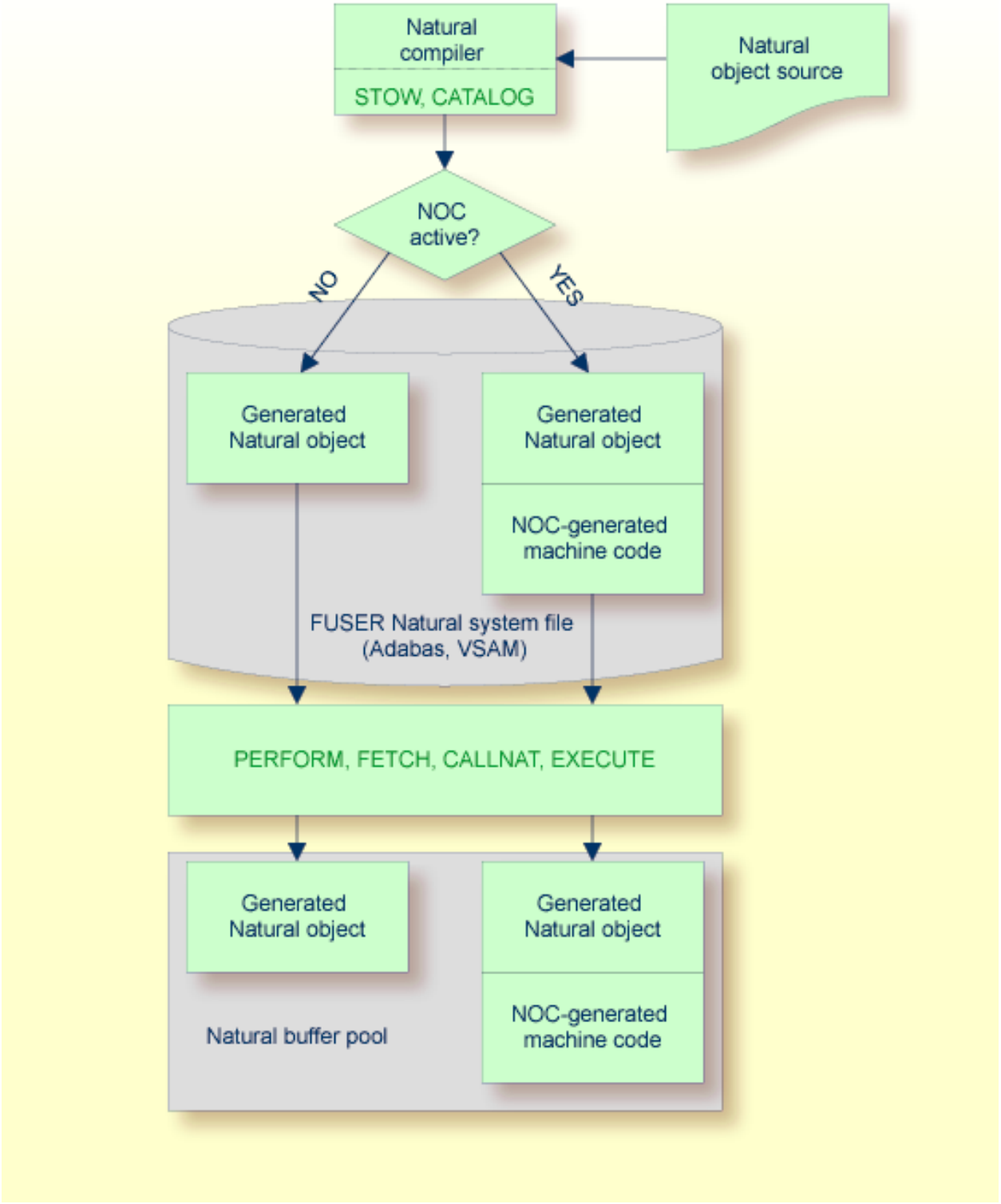
2 NOC - General Information

This section describes various aspects which should be taken into consideration when the Natural Optimizer Compiler is installed at your site. The information provided in this documentation helps you to make full use of the benefits offered by the Natural Optimizer Compiler.

Natural Nucleus Optimization

The Natural nucleus optimizes simple arithmetic, assignment, and comparison statements by translating parts of them into machine code. All programs are optimized automatically in this way.

The following graphic illustrates how the Natural Optimizer Compiler generates machine code when a Natural object is compiled or executed:


Natural compiler
STOW, CATALOG
Natural object source
NOC active?
NO
YES
Generated Natural object
Generated Natural object
NOC-generated machine code
FUSER Natural system file (Adabas, VSAM)
PERFORM, FETCH, CALLNAT, EXECUTE
Generated Natural object
Generated Natural object
NOC-generated machine code
Natural buffer pool

Natural Optimizer Compiler

The Natural Optimizer Compiler goes one step further than standard optimization. It compiles not only simple statements to machine code, but also complex statements and statement sequences.

The compiled code is further optimized as far as array range operations, field concatenation, and optimum base register assignment are concerned.

All statements (including arithmetic operations) optimized with the Natural Optimizer Compiler provide the same results as the same statements generated by standard Natural.

To activate the Natural Optimizer Compiler (see the relevant section), use the macro `NTOPT` in the Natural parameter module, the dynamic profile parameter `OPT`, the system command `NOCOPT`, or the `OPTIONS` statement.

All programs that are cataloged (`STOW` or `CATALOG` system command) with the Natural Optimizer Compiler activated are compiled to machine code. This will also result in the object code size of the programs being larger than usual, depending on how much of the program can be optimized.

A program executed with the `RUN` system command is compiled to machine code if the Natural Optimizer Compiler is activated with the system command `NOCOPT`, the macro `NTOPT` or the `OPTIONS` statement for all or part of the program.

To see if a program is suitable for compilation with the Natural Optimizer Compiler, use the `NOCSTAT` command as described in the relevant section.

Note: The dynamic recatalog feature (profile parameter `RECAT` set to `ON`) cannot be used with programs compiled to machine code.

To execute programs that have been compiled with the Natural Optimizer Compiler, it is not necessary that the Natural Optimizer Compiler is installed.

II Using the Optimizer Compiler - Overview

3 What is Compiled and What is Not

The Natural Optimizer Compiler is particularly effective for programs that contain a considerable amount of data manipulation, such as computation, transfer, and logical condition processing.

This section contains an overview of the statements which are compiled to machine code and those which are not compiled.

Note: The options the Natural Optimizer Compiler provides cannot be used for specifying statements to be optimized as described in the *Optimizer Options*.

Statements Compiled by the Natural Optimizer Compiler

The Natural Optimizer Compiler compiles the following statements to machine code:

- **Statements for Arithmetic and Data Movement Operations:**
  - ADD
  - ASSIGN
  - COMPRESS
  - COMPUTE
  - DIVIDE
  - EXAMINE, with the following clauses:
    - DIRECTION (with constant values only; that is FORWARD or BACKWARD),
    - GIVING NUMBER, GIVING POSITION (also concurrently),
    - GIVING LENGTH

    Example:

    ```
    EXAMINE #TEXT FOR #A GIVING NUMBER #NMB1
    EXAMINE #TEXT FOR #A GIVING POSITION #POSEX5
    EXAMINE #TEXT FOR #A GIVING LENGTH #LGHEX6 ↩
    ```

    Restrictions:

    - GIVING INDEX is not optimized.
    - *operand1* and *operand4* can be fix array occurences; that is, no ranges are admissible, for example:

```
EXAMINE #A(#J) FOR #B(#K)
```

- MOVE (ROUNDED, SUBSTRING, BY NAME, LEFT/RIGHT JUSTIFIED,)
- MOVE ALL
- MULTIPLY
- RESET
- SUBTRACT

- **Statements for Processing of Logical Conditions:**
  - IF
  - DECIDE FOR
  - DECIDE ON
- **Statements for Loop Execution:**

  FOR
- ESCAPE
- REPEAT

Statements that are Not Compiled

The Natural Optimizer Compiler *does not* compile the following statements:

- I/O statements (DISPLAY, WRITE, READ/WRITE WORK FILE).
- complex special statements such as SEPARATE.
- statements that pass control to another object such as FETCH, PERFORM, CALLNAT, CALL.
- statements that perform database access (READ, FIND, HISTOGRAM, GET, UPDATE, DELETE, END TRANSACTION, BACKOUT TRANSACTION)

4 NOCSTAT Command

For programs optimized with the Natural Optimizer Compiler, certain statements can be directly converted into machine code when cataloged. As a result, when executing the optimized objects with Natural at runtime, the performance can be improved considerably.

The NOCSTAT command analyses cataloged objects and provides statistical information to help decide whether program statements benefit from optimization with the Natural Optimizer Compiler and, if so, to what extent they can be optimized.

If a program is cataloged (STOW, CATALL), the Natural compiler generates an internal (pseudo) object code based on the statements in the source program. In most cases, one source statement is transformed into one pseudo-code instruction. However, for complex statements, such as FOR and REPEAT, several pseudo-code instructions are generated. The NOCSTAT analyses are based on the generated pseudo-code instructions. Therefore, the number of statements indicated in the statistical reports may exceed the number of statements in the source program.

Invoking NOCSTAT

To use the Natural NOCSTAT command

- Enter the direct command NOCSTAT.

  The main NOCSTAT screen is displayed:

```
16:41:00            ***** NATURAL NOCSTAT COMMAND *****            2017-11-24


Name ................. ________
Library ............... SAGTEST_

NOCable Objects only .. _

Output Report ......... X Statement Category
                        _ Statement Type
                        _ Code Profile

Output Destination .... X Screen
                        _ CSV to Work File
                        _ XML to Work File
                          with XSL ____________________________________

Progress Control ...... X
Download to PC ........ _


Command ===>


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                 Canc
```

To obtain field-specific help information, either enter a question mark in the relevant field and press ENTER, or place the cursor in the field and press PF1. Press PF3 to exit NOCSTAT.

Generating Reports

You can generate statistical reports for a single program or a set of programs. If you analyze more than one program at a time, the reports are produced in series. When you have finished looking at one report, press ENTER to view the next report.

The main NOCSTAT menu provides the following options:

Field	Explanation	
Name	Enter a name or a range of names to specify the program(s) you want to examine:	
	value is any combination of one or more characters.	
	value	Single program.
	*	All programs.
	*value**	All programs whose names begin with *value*.
	value>	All programs whose names are greater/equal *value*.

Field	Explanation	
	value<	All programs whose names are less/equal *value*.
Library	Enter the name of a library or specify a range; the same applies as described for the Name field above. The current library is the default.	
NOCable Objects only	Mark this option to exclude programs already compiled with the Natural Optimizer Compiler. Otherwise, the NOCSTAT command selects all Natural programs specified in the Name and Library fields by default, including NOC-compiled programs.	
Output Report	Mark any of the options to select statements by category, type or code profile. See ***Statement Category***, ***Statement Type*** and ***Code Profile*** below.	
Output Destination	Mark any of the following options to determine the output format and destination:	
	Screen	Displays the report on the screen or writes the report data to Print File 7 if Download to PC is selected for processing.
	CSV to Work File	Generates spreadsheets with comma-separated values. The report data is written to either of the following files: 1. Work File 7 if running online and Download to PC is selected. 2. Work File 1 in all other cases. Use the file extension .csv to write the work file directly to your PC for further processing. You can only route reports to a PC if Entire Connection is installed.
	XML to Work File	Generates XML documents. The report data is written to either of the following files: 1. Work File 7 if running online and Download to PC is selected. 2. Work File 1 in all other cases. Use the file extension .txt to write the work file data directly to your PC and change the file name afterwards to the extension .xml for further processing. You can only route reports to a PC if Entire Connection is installed. If a value is entered in the field with XSL, a processing instruction is added at the top of the XML output document:

Field	Explanation	

		`<?xml-stylesheet type="text/xsl" href="` *value* `"?>`
		
		The *value* entered should be the absolute or relative URL of the style sheet, for example:
		
		`nocstat.xsl`
		
		or
		
		`http://natural.software-ag.de/nocstat.xsl`
		
		The processing instruction causes the document to be transformed according to the given style sheet when it is viewed by an XSLT-capable browser or transformed by a batch XSLT run. A typical use of this feature is to convert the output XML to an HTML page.
		
		There are two XSLT style sheets delivered with Natural as text objects `NOCSTLS1` and `NOCSTLS2` in the Natural library `SYSEXUEX` in the `FNAT` system file.
		
		`NOCSTLS1` provides formatting instructions for report type ***Statement Category***, `NOCSTLS2` for report type ***Statement Type*** as described below.
		
		Download the style sheets with file extension `.xsl` to the same directory in which the XML work files are stored.

`Progress Control`	Only applies in an online environment and if one of the following options is selected for processing: 1. `CSV to Work File`, 2. `XML to Work File`, 3. `Download to PC`. If one of these options is selected, a brief message appears for each program listed in the generated report.	
`Download to PC`	Only applies if Entire Connection is installed, and if you run online with Print/Work File 7 defined as the PC file (see the `WORK` and `PRINT` profile parameters). Download report output data with Entire Connection to a PC by using either of the following: 1. Print File 7 for the `Screen` destination. 2. Work File 7 for CSV or XML output.	

Report Formats

You can choose between three output formats described below to display the statistics NOCSTAT provides for the statements analyzed. Different report layouts are produced for programs already optimized with the Natural Optimizer Compiler and for programs to be considered for optimization. The example reports below show the difference. Press PF3to interrupt report processing and return to the NOCSTAT menu.

Below is information on:

- Statement Category
- Statement Type
- Code Profile

Statement Category

The statistical report generated with the option Statement Category lists various categories of statements with the corresponding number of occurrences and the total number of statements already optimized or suitable for optimization, depending on whether or not the program was optimized with the Natural Optimizer Compiler.

Example of Program without NOC Optimization:

```
11:49:46              ***** NATURAL NOCSTAT COMMAND *****              2017-05-29
                 Library SAGTEST  Name NOCTEST1 Type Program


                       No NOC      NOCable
                      --------     --------
      Database Loop:         0            0
    Database Simple:         0            0
    SORT / WORK I/O:         0            0
       FOR / REPEAT:         0            1
   Screen / Printer:         1            0
String Manipulation:         6           34
    Arith / Logical:         0          996
      Program Calls:        20            0
   Control Transfer:         2          182
        Block Start:         1            0
     Set Environment:         7            0
   System Functions:          2            0
      Miscellaneous:         0            1

                 Total Statements:     1254
                  NOC optimizable:     1214  ( Ratio:  96 % )
                  Longest NOC Run:      216  Statements
```

Example of NOC-Optimized Program:

```
11:51:25            ***** NATURAL NOCSTAT COMMAND *****            2017-05-29
               Library SAGTEST  Name NOCTEST1 Type Program

     MCG Options: (ON,OVFLW,INDX,MIX,IO)


      Database Loop:        0
    Database Simple:        0
    SORT / WORK I/O:        0
       FOR / REPEAT:        0
   Screen / Printer:        1
String Manipulation:       36
    Arith / Logical:        0
      Program Calls:       20
   Control Transfer:        2
        Block Start:        1
    Set Environment:        7
   System Functions:        2
      Miscellaneous:        1

   Total Statements:     1255
      NOC optimized:     1185  ( Ratio:  94 % )
    Longest NOC Run:      136  Statements
```

Report Columns and Fields:

Column	Explanation
No NOC	Statements not suitable for optimization.
NOCable	Statements suitable for optimization. **Note:** The number of NOCable statements is only a reasonable assumption but cannot be considered an absolutely reliable value. This is because the NOCSTAT command cannot perform all analytical queries and, occassionally, very complex code investigations that definitely decide whether a statement can be optimized with the Natural Optimizer Compiler.
Field	
Database Loop	The number of database statements that generate a processing loop, such as FIND and READ.
Database Simple	Database statements that do not generate a processing loop, such as STORE, UPDATE, DELETE and GET.
SORT / WORK I/O	SORT and work file statements.
FOR / REPEAT	Statements generating loops.
Screen / Printer	Screen and printer I/O, such as WRITE, DISPLAY and INPUT.
String Manipulation	String statements, such as EXAMINE and COMPRESS.

Column	Explanation
Arith / Logical	Arithmetic and logical statements, such as MOVE, COMPUTE and IF.
Program Calls	Transfer of control to a subroutine or subprogram, such as PERFORM, CALLNAT and FETCH.
Control Transfer	Jumps within the program, such as ESCAPE BOTTOM, FOR and REPEAT loops.
Block Start	Non-executed statements that demarcate code blocks, such as DEFINE SUBROUTINE and AT END. These statements are never optimized because they are never executed.
Set Environment	Statements that set the environment, such as SET CONTROL, SET GLOBALS and SET KEY.
System Functions	Statements, such as TOTAL, SUM, COUNT, MAX, MIN and *COUNT.
Miscellaneous	Pseudo-code statements not relevant for optimization and, therefore, ignored by the NOC.
Totals	
Total Statements	The total number of statements found in the program. This number may not correspond to the actual source statements as described in the introduction to NOCSTAT command above.
NOC optimized	For an optimized program, these are the actual pseudo-code statements (as described in the introduction to NOCSTAT command above) that have been NOC-optimized to machine code.
NOC optimizable	For non-optimized programs, this is the possible number of statements that could be optimized. The figure may be slightly higher than the actual number, since certain factors are not considered in the NOCSTAT program. For example, a SUBSTRING statement that has more than four arrays will be indicated as “optimizable” though it will not be optimized.
Ratio	Relation between Total Statements and NOC-optimized statements or Total Statements and NOC-optimizable statements in percent.
Longest NOC Run	NOC-optimized program: The number of contiguous optimized statements - the fewer fragment sequences, the better the performance.
	Non-optimized program: The number of contiguous statements to be expected if the program were optimized.

Statement Type

The statistical report generated with the option `Statement Type` lists single statements with the corresponding number of occurrences and the NOC coding generated for optimized objects.

Example of Program without NOC Optimization:

```
12:29:23            ***** NATURAL NOCSTAT COMMAND *****            2017-05-29
               Library SAGTEST  Name NOCTEST1 Type Program

       Statement            No NOC   NOCable
------------------------   --------  --------
MOVE/COMPUTE/ASSIGN               0       615
EXAMINE                           6         0
SEPARATE                          0        30
COMPRESS                          0         4
MOVE TO SYSTEM FUNCTION           2         0
CALLNAT/PERFORM EXTERNAL         17         0
MOVE EDITED                       1         0
ELSE/CLOSE LOOP                   0       182
ON ERROR                          1         0
END                               1         0
STOP                              1         0
IF                                0        51
IF IN REPEAT UNTIL                0         1
REPEAT                            0         1
RESET                             0        74
IF                                0       255
FETCH                             3         0
IGNORE                            0         1
STACK TOP CMD/DATA                2         0
MCG OPTIONS                       1         0
OPTIONS                           1         0
SET CONTROL                       4         0
```

Example of NOC-Optimized Program:

```
12:31:30             ***** NATURAL NOCSTAT COMMAND *****             2017-05-29
                Library SAGTEST  Name NOCTEST1 Type Program
     MCG Options: (ON,OVFLW,INDX,MIX,IO)
       Statement            Number
-------------------------   --------
EXAMINE                          6
SEPARATE                        30
MOVE TO SYSTEM FUNCTION          2
CALLNAT/PERFORM EXTERNAL        17
MOVE EDITED                      1
NOC CODE                      1183
ON ERROR                         1
END                              1
STOP                             1
FETCH                            3
IGNORE                           1
STACK TOP CMD/DATA               2
MCG OPTIONS                      2
OPTIONS                          1
SET CONTROL                      4
```

Code Profile

The statistical report generated with the option `Code Profile` displays contiguous sequences of statements grouped by categories in a source program suitable for optimization, or lists the NOC coding generated for an optimized program. Occurrences are highlighted.

Example of Program without NOC Optimization:

```
12:38:52             ***** NATURAL NOCSTAT COMMAND *****             2017-05-29
               Library SAGTEST  Name NOCTEST1 Type Program

Line   Statement
-----  -------------------------
0000   ON ERROR
0000   MCG OPTIONS
0000   OPTIONS
0295   CALLNAT/PERFORM EXTERNAL
0295   MOVE/COMPUTE/ASSIGN       <-- NOCable
0295   MOVE/COMPUTE/ASSIGN       <-- NOCable
0295   MOVE/COMPUTE/ASSIGN       <-- NOCable
0295   MOVE/COMPUTE/ASSIGN       <-- NOCable
0740   MOVE/COMPUTE/ASSIGN       <-- NOCable
0745   IF                        <-- NOCable
0750   MOVE/COMPUTE/ASSIGN       <-- NOCable
0755   MOVE/COMPUTE/ASSIGN       <-- NOCable
0760   CALLNAT/PERFORM EXTERNAL
0765   IF                        <-- NOCable
0770   MOVE/COMPUTE/ASSIGN       <-- NOCable
0775   ELSE                      <-- NOCable
0780   MOVE/COMPUTE/ASSIGN       <-- NOCable
0810   RESET                     <-- NOCable
MORE
```

Example of NOC-Optimized Program:

```
12:39:47            ***** NATURAL NOCSTAT COMMAND *****            2017-05-29
               Library SAGTEST  Name NOCTEST1 Type Program

Line   Statement
-----  -------------------------
0000   MCG OPTIONS
0005   MCG OPTIONS
0000   OPTIONS
0295   CALLNAT/PERFORM EXTERNAL
0295   NOC CODE
0295   NOC CODE
0295   NOC CODE
0295   NOC CODE
0740   NOC CODE
0745   NOC CODE
0750   NOC CODE
0755   NOC CODE
0760   CALLNAT/PERFORM EXTERNAL
0765   NOC CODE
0770   NOC CODE
0775   NOC CODE
0780   NOC CODE
0810   NOC CODE
MORE
```

Batch Execution

Below are job examples for processing `NOCSTAT` reports in batch mode to create a CSV work file. After job execution, the work files generated can be transferred from host to PC for further processing with standard transfer tools.

Example Job z/OS:

```
//NOCBATCH JOB  (NOC,,,30),CLASS=K,MSGCLASS=X                          00000100
//NATEX EXEC PGM=NATvrsBA,REGION=6200K,PARM=('IM=D')                   00000200
//STEPLIB  DD DISP=SHR,DSN=TESTNAT.LOAD                                00000300
//CMPRINT  DD SYSOUT=X                                                 00000400
//CMWKF01  DD DSN='NOC.NOCSTAT.OUT',DISP=(NEW,CATLG),                  00000500
            SPACE=(CYL,(1,1)),UNIT=SYSDA,VOL=SER=SAG001                00000600
//SYSOUT   DD SYSOUT=X                                                 00000700
//CMSYNIN  DD *                                                        00000800
NOCSTAT                                                                00000900
*,library,,X,,,,X                                                      00001000
.                                                                      00001100
```

```
FIN                                                                          00001200
/*                                                                           00001300
```

Example Job z/VSE:

```
* $$ JOB JNM=NOCTST,CLASS=5,DISP=D
* $$ LST CLASS=Q,DISP=D
// JOB NOCTST
// ASSGN SYS001,DISK,VOL=xxxxxx,SHR
// DLBL CMWKF01,'NOCSTAT.FILE.ONE',0
// EXTENT SYS001,xxxxxx,1,0,1,150
// EXEC NATvrsBA,SIZE=NATvrsBA,PARM='SYSRDR'
IM=D,OBJIN=R
/*
ADARUN DBID=185
/*
NOCSTAT
*,library,,X,,,,X
.
FIN
/*
/&
```

Example Job BS2000:

```
/.BAT234 LOGON  NAT,1
/        SYSFILE SYSOUT=NATvrs.OUT
/        SYSFILE SYSLST=NATvrs.LST
/SKIP   .NOP000
================================================================================
      NAME   : E.NATvrs          S T A R T  B A T C H   N A T U R A L
================================================================================
/.NOP000 REMARK
/        OPTION  DUMP=YES,MSG=FL
/        FILE    NOCSTAT.OUT,LINK=W01
/        FILE    ADAUSER  ,LINK=DDCARD
/        FILE    $SAG.ADAvrs.MOD       ,LINK=BLSLIB00
/        SYSFILE TASKLIB=MODvrs
/        SYSFILE SYSDTA=(SYSCMD)
/        FILE    NATvrs.CMPRMIN,LINK=CMPRMIN
/        DCLJV   NATJV1,LINK=*NATB2JV
/        FILE    $NAT.ADALNK.PARMS,LINK=DDLNKPAR
/        REMARK  %%%%%%%%%%  BATCH-PHASE   %%%%%%%%%%%%%%
/        EXEC    NATvrs
NOCSTAT
*,library,,X,,,,X
.
FIN
```

5 Displaying the Size of the Machine Code

With the Natural system command `LIST DIRECTORY`, you can see whether a program has been compiled to machine code and also the size of the machine code.

To list compiled programs

- Enter the Natural system command

```
LIST DIR object-name
```

The directory information for the specified object will be displayed, showing at the bottom of the screen the size of the machine code, the `OPT` parameters used for the compilation and the Natural Optimizer Compiler version under which the program was cataloged.

Further details of the `LIST` command are provided in the *System Commands* documentation.

6 Optimizer Usage Examples

The examples below illustrate when to use the Natural Optimizer Compiler to the best advantage and to give an indication of its power:

Example 1 - No Improvement

Nothing would be gained by using the Natural Optimizer Compiler for the following program, since it contains a statement that performs database access and an I/O statement (see *Statements that are Not Compiled*):

```
DEFINE DATA LOCAL
  1 EMPLOYEES VIEW OF EMPLOYEES
    2 JOB-TITLE
    2 BIRTH
    2 NAME
  END-DEFINE
  FIND EMPLOYEES WITH JOB-TITLE = 'PROGRAMMER' OR = 'ANALYST'
                              OR = 'PROGRAMMER/ANALYST'
                              OR = 'SYSTEM ANALYST'
    DISPLAY JOB-TITLE BIRTH NAME
  END-FIND
  END
```

Example 2 - Considerable Improvement

If the following program is compiled with the Natural Optimizer Compiler, you will see a performance improvement of approximately 30 % (that is a 30 % reduction in CPU load). The program performs a statistical analysis of the age of IT-employees. Optimized statements are indicated in boldface.

In this example, the Natural Optimizer Compiler increases the object size by 20.5 %, due to 952 bytes of additional machine code:

Profile Parameter Setting	Size in Buffer Pool	Size of Machine Code Generated by NOC
OPT=NODBG	5768	952
OPT=OFF	4784	0

```
 DEFINE DATA
 LOCAL
 1 EMPLOY VIEW OF EMPLOYEES
   2 JOB-TITLE     (A25)
   2 BIRTH         (D)
 1 I               (I1)  INIT <1>
 1 CDATE           (D)
 1 NUMB            (N4)
 1 SUMM            (P7.2)
 1 SQUARE          (F8)
 1 DEVI            (F8)
 1 DEVIATION       (N3.4)
 1 MEAN            (P2.3)
 1 AGEDIS          (F8/1:70)
 1 AGEMAX          (F8)
 1 AGEH            (P3)
 1 AGE             (P3)
 1 AGEDAYS         (P15)
 1 LINE            (A71/1:20)
 1 REDEFINE LINE
   2 POINTS        (A1/1:20,0:70)
 END-DEFINE
 *
 MOVE *DATX TO CDATE
 *
 FIND EMPLOY WITH JOB-TITLE = 'PROGRAMMER' OR = 'ANALYST'
   OR = 'PROGRAMMER/ANALYST' OR = 'SYSTEM ANALYST'
AGEDAYS:= CDATE - BIRTH
   AGE:=AGEDAYS / 365
   ADD 1 TO AGEDIS(AGE)                  /* DISTRIBUTION
   ADD 1 TO NUMB
   ADD AGE TO SUMM
   COMPUTE SQUARE = SQUARE + AGE * AGE
 END-FIND
 *
 ***********************************************************************
 * COMPUTE ESTIMATES
 ***********************************************************************
 *
 COMPUTE DEVI = NUMB * SQUARE / (SUMM * SUMM) - 1
 COMPUTE DEVIATION = SQRT(DEVI)
 COMPUTE MEAN = SUMM / NUMB
 *
 ***********************************************************************
 * GRAPHIC DISPLAY
 ***********************************************************************
 *
 FOR I 1 70
   IF AGEDIS(I) > AGEMAX MOVE AGEDIS(I) TO AGEMAX
   END-IF
 END-FOR
```

```
FOR I 1 70
  COMPUTE AGEDIS(I) = AGEDIS(I) * 20 / AGEMAX
END-FOR
FOR I 1 70
  COMPUTE AGEH = 21 - AGEDIS(I)
  IF AGEH < 21 MOVE '*' TO POINTS(AGEH:20,I)
  END-IF
END-FOR
*
************************************************************
* COMPLETE GRAPHIC DISPLAY
************************************************************
*
MOVE '!' TO POINTS(*,0)
WRITE TITLE LEFT
 AGEMAX(EM=999)  20X  'DISTRIBUTION OF IT-EMPLOYEES BY AGE'
WRITE NOTITLE NOHDR
LINE(*) /
'0--------10--------20--------30--------40--------50--------60--------'
/  'MEAN='
```

Examples 3 and 4 - CPU Usage

The following program illustrates the difference in CPU usage, depending on the options you select when compiling the program. The table below lists the CPU usage in seconds and percent. The figures provided in the table were determined during a test run in an IBM z/OS environment. They can only serve as general orientation, since absolute values vary depending on the hardware applied.

```
DEFINE DATA LOCAL
1 #I1         (I4) INIT <1>
1 #I2         (I4) INIT <2>
1 #J1         (I4) INIT <3>
1 #J2         (I4) INIT <4>
1 #F          (I4)
1 #ARR1       (N7/10,5)
1 #ARR2       (N5/10,5)
END-DEFINE
*
FOR #F = 1 TO 1000000
  MOVE #ARR1(#I1,#I2) TO #ARR2(#J1,#J2)
END-FOR
*
END
```

Option	CPU seconds	CPU percentage
OFF	8.78	100
ON	0.63	7.18
INDX	0.85	9.68
OVFLW	1.71	19.48
INDX,OVFLW	2.00	22.78
INDX,OVFLW,NODBG	1.61	18.34
INDX,OVFLW,NODBG,NOSGNTR	1.61	18.34
NODBG	0.44	5.01
NOSGNTR	0.63	7.18
NODBG,NOSGNTR	0.44	5.01

```
DEFINE DATA LOCAL
1 #I1         (P7) INIT <1>
1 #I2         (P7) INIT <2>
1 #J1         (N7) INIT <3>
1 #J2         (N7) INIT <4>
1 #K1         (I4) INIT <5>
1 #K2         (I4) INIT <6>
1 #F          (I4)
1 #FIELD1     (P5)
1 #FIELD2     (N5)
1 #FIELD3     (I2)
END-DEFINE
*
FOR #F = 1 TO 500000
*
  #FIELD1:= #I1 - #I2 + (13 * 10 / 5)
  #FIELD2:= #J1 - #J2 + (13 * 10 / 5)
  #FIELD3:= #K1 - #K2 + (13 * 10 / 5)
*
END-FOR
*
END
```

Option	CPU seconds	CPU percentage
OFF	18.61	100.00
ON	4.95	26.60
INDX	4.95	26.60
OVFLW	5.38	28.91
INDX,OVFLW	5.38	28.91
INDX,OVFLW,NODBG	5.26	28.26
INDX,OVFLW,NODBG,NOSGNTR	5.09	27.35

Option	CPU seconds	CPU percentage
NODBG	4.79	25.74
NOSGNTR	4.81	25.85
NODBG,NOSGNTR	4.63	24.88
NODBG,NOSGNTR,ZD=OFF	4.51	24.23
NODBG,NOSGNTR,ZD=OFF,SIGNCHCK=OFF	4.41	23.70

III

7 Activating the Optimizer Compiler

To activate the Natural Optimizer Compiler, use one of the methods described in the following sections, where the first alternative is the most static one and the last alternative the most dynamic one.

All alternatives use the Optimizer options as described in the section *Optimizer Options*. Using these options you can control how and when machine code is generated, what tracing options are to be used and what the target architecture will be. The Optimizer options are the only control mechanism for the Natural Optimizer Compiler.

Macro NTOPT

With the macro `NTOPT` in the Natural parameter module, you can activate the Natural Optimizer Compiler statically for a linked Natural nucleus. Every time this Natural nucleus is started, the same Optimizer options are used again.

Example 1:

```
        NTOPT 'INDX,OVFLW,ZD=OFF'
```

Example 2:

```
        NTOPT 'INDX,OVFLW,ZD=OFF,TRGPT',                                *
              'TRSTMT,OPTLEV03'
```

Note the continuation character "*" in column 72.

See the section *Optimizer Options* for an explanation of the options setting used.

Dynamic Profile Parameter OPT

When starting a Natural session, you can dynamically activate the Optimizer Compiler by specifying the Natural profile parameter `OPT`. As a synonym for `OPT`, you can use `MCG`. The specification of the parameter module is overwritten. The options are only valid for the current session.

Example:

```
OPT=(INDX,OVFLW,ZD=OFF)
```

or

```
MCG=(INDX,OVFLW,ZD=OFF)
```

See the section *Optimizer Options* for an explanation of the option setting used.

System Command NOCOPT

When you have started a Natural session, you can invoke the Optimizer command screen with the Natural system command `NOCOPT`. The screen monitors the current setting of the Natural Optimizer Compiler options as they were specified during Natural startup. You can now modify the setting online.

The updated parameter setting is only valid for the current session.

Natural Statement OPTIONS

The `MCG` parameter of the Natural compiler statement `OPTIONS` provides the most flexible and powerful control over machine code generation, since different options can be set for individual statements in a program. So, within one Natural program, the Natural Optimizer Compiler can be activated and deactivated several times to enclose ranges of statements with different options settings.

Example

```
OPTIONS MCG=(OVFLW,INDX,ZD=OFF)
```

or

```
OPTIONS MCG=OVFLW,INDX,ZD=OFF
```

The options string of the MCG parameter may start with a plus (+) or minus (-) sign, indicating that the values of options not mentioned should be left unaltered, and only the options present should be set (+) or reset (-), for example:

Example:

```
   OPTIONS MCG=+PGEN         /* turns tracing on

  (statements to be traced)

 OPTIONS MCG=-PGEN         /* turns tracing off
```

If the string starts with anything other than "+" or "-", all options are reset before the string is parsed.

Note: The Natural statement OPTIONS also provides other Natural compiler parameters than MCG.

See the section ***Optimizer Options*** for an explanation of the options setting used.

8 Optimizer Options

When the Natural Optimizer has been activated, you can specify checks by setting the options explained in this section.

The options cannot be used for specifying statements to be optimized.

List of Options

The following table lists and describes the Natural Optimizer Compiler options. Default values are underlined (this is the value that will be assumed if the option is not present).

A Natural Optimizer Compiler option consists of a string surrounded by brackets or single quotation marks (except in the Natural `OPTIONS` statement), with options separated by commas. Some options have values, while the very existence of some options in the option string is sufficient to modify the environment.

The following rules apply:

- Optional clauses are surrounded by square brackets [ ].
- Choices are surrounded by curly braces { }.
- Each choice is separated by vertical lines "`|`".
- Only one of these choices can be specified;

  `ON` is equivalent to `Y` (Yes),

  `OFF` to `N` (No).
- Options specified without the optional clause `ON` or `OFF` (if applicable), or their equivalent values, are interpreted as set to `ON`. For example, `OVFLW` is identical to `OVFLW=ON`.
- Except for the option `OFF`, any specified option switches on optimizing (as if `ON` was specified) and the default values apply. For example, `INDEX` is identical to `ON,INDEX`.

Option	Explanation
`ABEND`	Forces the Natural Optimizer Compiler to generate code which causes Natural to be abnormally terminated immediately when the `ABEND` option is encountered by the Natural Optimizer Compiler during compilation. The option must appear by itself or it will be ignored. Other parameters are not changed or reset by this option. This option can be useful for debugging purposes.
`ARCH`	Specifies the architecture level to be used for code generation: see ***ARCH Option*** in the following section.
`ARROPT`	Specifies the generation to be used for array assignments of the type `A(*):=`*`scalar`*. See ***ARROPT Option*** in the following section.
`CACHE[={ON| OFF |Y|N}]`	Switches variable caching on or off. See also ***Variable Caching*** in the section *Performance Considerations*.

Option	Explanation
CPU= /370	Specifies the target architecture.
DIGTCHCK[={ON| OFF |Y|N}]	Specifies whether the digits of packed and unpacked numeric fields (formats P and N) are to be checked when moving to another variable of the same type and precision. For example, if DIGTCHCK is ON and an unpacked numeric variable (format N) contains an invalid digit, such as X'FA', moving to another unpacked numeric variable with the same precision will generate a SOC7 (or NAT0954) error. If DIGTCHCK is OFF, no error is generated but the generated code is much faster.
ERRDUMP[={ON| OFF |Y|N}]	Specifies whether NOC should abend if an error condition is detected during the compile phase. This is useful for debugging the Natural Optimizer Compiler itself.
INDEX[={ON| OFF |Y|N}]	Specifies whether array indexes will be checked for out-of-bound values in the optimized code. See also the following ***Note***.
INDX[={ON| OFF |Y|N}]	Specifies whether array indexes will be checked for out-of-bound values in the optimized code. Additionally, RANGE will be set on. Therefore, this option is equivalent to INDEX=ON,RANGE=ON. See also the following ***Note***.
IO[={ON| OFF|Y|N}]	Provided for compatibility reasons only. No effect.
LOOPS[={ON| OFF |Y|N}]	Provided for compatibility reasons only. No effect.
MIX[={ON| OFF |Y|N}]	Provided for compatibility reasons only. No effect.
NODBG[={ON| OFF|Y|N}]	If NODBG=OFF/N (default), the Natural Debugger can be used to debug optimized code (then, additional code is generated to check whether TEST mode has been set on). If NODBG=ON/Y, less code will be generated, the program will run faster and consume less CPU time. On the other hand, the functionality of the Natural Debugger will be limited, because the Natural Debugger might not receive control for optimized statements. See also NODBG in the section *Performance Considerations*.
NOSGNTR[={ON| OFF |Y|N}]	Applies to packed numbers only. If NOSGNTR=OFF (default), signs of positive packed numbers which are the result of an arithmetic operation or the target of an assignment are set according to the COMPOPT parameter PSIGNF. If NOSGNTR=ON, the signs resulting from execution of the generated machine instruction are left unchanged. See also the section ***Influence of other Natural Parameters***.
ON	Switches on optimizing. If no additional option is specified, the default value defined for each option is in effect. As indicated in the following ***Note***, this may

Option	Explanation
	cause unintended results, in particular regarding the options INDEX, INDX, OVFLW, and RANGE.
OFF	Switches off optimizing.
OPTLEV={ 2|3}	Specifies optimization level - roughly equivalent to the number of passes through the program. OPTLEV=3 is useful when PGEN is specified, since some branch targets cannot be determined during the first pass and PGEN output is made during the last pass. Thus, some values may be shown improperly.
OVFLW[={ON| OFF |Y|N}]	Specifies whether checks for overflow in arithmetic operations or assignments will be included in the optimized code. See also the following ***Note***.
PGEN[={ON| OFF |Y|N}]	Specifies whether a disassembly of the optimized code should be output. This option also enables all other tracing options: see ***PGEN Option*** in the following section.
RANGE[={ON| OFF |Y|N}]	Specifies whether range checks will be performed in operations with arrays. This ensures that array ranges will have an equal number of elements in corresponding dimensions of all operands. See also the following ***Note***.
SIGNCHCK[={ ON|OFF|Y|N}]	Specifies whether the result of a multiplication with a packed or unpacked numeric multiplier should be checked for a negative zero. If zero is multiplied by a negative number, the MP machine instruction generates a negative zero result. If SIGNCHCK is on, this negative zero is converted to a positive zero. The check for a negative zero is done for every multiplication with a packed or unpacked numeric multiplier.
TRENTRY	For internal use by Software AG only. Do not change the setting of this parameter.
UNICC	Specifies whether optimized code is generated for IF and DECIDE statements with Unicode operands: see ***UNICC Option*** in the following section.
ZD[={ ON|OFF|Y|N}]	Specifies whether divisors should be checked for zero. If this option is specified, then code is inserted, so that the program behaves according to the ZD profile parameter of Natural, that is, Natural error NAT1302 is issued or the result is zero. If this option is not specified, Natural error NAT0954 occurs if the divisor is zero. See also *ZD - Zero-Division Check* in the Natural *Parameter Reference* documentation.

Note for INDEX, INDX, OVFLW and RANGE:

If the option INDEX, INDX, OVFLW or RANGE is set, extra instructions are added to the generated code to detect data overflow and index-out-of-range situations should they occur during program execution. Although the use of these options slightly increases the generated code, we recommend

to use them to guarantee that erroneous programs are detected and cannot lead to unpredictable results, storage corruptions or abnormal program terminations.

Example of INDEX and OVFLW

```
DEFINE DATA LOCAL
...
1 P1 (P1/9)
...
1 P3 (P3/9)
...
1 I (I4)
1 J (I4)
1 K (I4)
1 L (I4)
END-DEFINE
...
P1(I:J) := P3(K:L)
...
END
```

Explanation of Example

With `INDX=ON` or `INDEX=ON` set, code is generated to verify that `I`, `J`, `K` and `L` are within the ranges defined for `P1` and `P3` respectively.

With `INDX=ON` or `RANGE=ON` set, code is generated to verify that `I:J` and `K:L` denote ranges of the same length.

With `OVFLW=ON` set, code is generated to verify that the value of `P3` fits into the corresponding `P1` variable.

For example: Value `100` would cause an overflow here.

Example Error Situation:

If one of the occurrences of `P3` contains the value `100`, with `OVFLW=OFF` set, the value assigned to the corresponding `P1` occurrence will be zero. If the index variable `I` is zero or greater than `9`, with `INDX=OFF` set, storage areas that do not belong to Array `P1` will be corrupted. If these options (`OVFLW` and `INDX`) are set to `ON`, a Natural error occurs like it does in standard Natural runtime.

For the NOC option specified above, additional code is generated. However, this is well compensated for by the advantage of a check that, for example, protects against hard-to-debug errors. Undetected errors can, of course, lead to unpredictable results.

Optimum Code Generation

To assure that the least amount of code is generated and thus achieve optimum performance, use:

```
OPT='NODBG,NOSGNTR,SIGNCHCK=OFF,ZD=OFF'
```

However, only apply this setting to objects that have been thoroughly debugged; see also *Note for INDEX, INDX, OVFLW and RANGE*.

ARROPT Option

The `ARROPT` option determines the generation algorithm to be used for array assignments of the type `A(*):=scalar`.

Valid values for `ARROPT` are:

Value	Explanation
`ON`	Recommended setting for arrays with a minimum of 50 occurrences. `ON` is the default setting.
`OFF`	Recommended setting for arrays with less than 50 occurrences.

PGEN Option

The `PGEN` option causes the Natural Optimizer Compiler to output the generated code and internal Natural structures. Thus, code and structures can be examined, for example, for bug fixing, performance review and support issues.

An understanding of IBM's /370 assembler is required to interpret the results produced by the `PGEN` option.

We recommend that you use this option with the assistance of your local Software AG representative.

- Setting PGEN
- Sub-Options of the PGEN Option
- Output of the PGEN Option

- Working with the PGEN Output

Setting PGEN

To use the PGEN facility, set the PGEN option when activating on the Optimizer Compiler.

Since the buffer is kept in memory, it is possible that the user thread will not be big enough to hold the trace information. In this case, try setting PGEN on only for the portion of the program which is to be traced, for example:

OPTIONS MCG=(PGEN=ON,TRGPT=ON) or OPTIONS MCG=+PGEN,TRGPT	Turns tracing on, including tracing of the GPT entries
OPTIONS MCG=(PGEN=OFF) or OPTIONS MCG=-PGEN	Turns tracing off

Various options affect the content of the output. The basic PGEN option causes a formatted listing of Natural source lines and a disassembly of the corresponding code to be generated and kept in memory for extraction by the NOCSHOW utility as described below, under ***Output of the PGEN Option***.

The TRSTMT, TRGPT, TRMPT and TRVDT options cause hex dumps of internal data structures associated with each line to be output.

The TRBASES and TRCACHE options cause information on base registers and cache variables to be printed out.

Sub-Options of the PGEN Option

The following table describes the options when PGEN=ON. For an explanation of the syntax used see the introduction to ***List of Options*** above.

Option	Explanation
LPP={5|..| 55 |..|255}	Lines-per-page for the trace output, only used when TREXT=ON.
NOsrcE[={ON| OFF |Y|N}]	If NOsrcE=OFF, the Natural source statement is included in the output.
TRACELEV={ 0 |..|255}	Specifies the trace level. Each bit in this one byte value specifies a buffer type to trace; these bits can be set on by using the TR*xxx* options as well.
TRBASES[={ON| OFF |Y|N}]	Specifies whether base register allocations are traced.
TRCACHE[={ON| OFF|Y|N}]	Specifies whether CACHE entries are traced.
TREXT[={ON| OFF |Y|N}]	If TREXT=ON, trace is directed to the user exit NOCPRINT as described below.
TRGPT[={ON| OFF |Y|N}]	Specifies whether GPT entries are traced.
TRMPT[=ON| OFF |Y|N}]	Specifies whether MPT entries are traced.

Option	Explanation
TRSTMT[={ON| OFF |Y|N}]	Specifies whether STMT entries are traced.
TRVDT[={ON| OFF |Y|N}]	Specifies whether VDT entries are traced.

See also the examples below.

Output of the PGEN Option

There are two places to where the Natural Optimizer Compiler can direct the output of PGEN:

- Internal Buffer
- User Exit NOCPRINT

Internal Buffer

The contents of this buffer is overwritten each time a CHECK, CAT, STOW or RUN command is executed. A system utility NOCSHOW is provided whereby the contents of this buffer can be viewed, searched or printed.

To invoke the NOCSHOW utility

- Enter the direct command NOCSHOW after a CHECK, STOW, CAT or RUN where the Natural Optimizer Compiler has been active.

The following PF keys are available on the screen:

PF Key	Function
PF2	Position to top of output
PF4	Position one line backward
PF5	Position one line forward
PF6	Print to report (1)
PF7	Position one page backward
PF8	Position one page forward
PF9	Print via Entire Connection to report (7)
PF10	Scan for text string
PF11	Repeat scan

User Exit NOCPRINT

If TREXT=ON is specified, the Natural Optimizer Compiler passes every output line to the user exit NOCPRINT instead of adding it to the trace buffer.

NOCPRINT is invoked following normal OS register conventions. Register 1 points to a full word containing the address of the 81 byte print line with ANSI carriage control characters in position 1. Register 13 points to an area of 18*4 bytes which may be used as a save area. Register 14 contains the return address and Register 15 contains the entry address of NOCPRINT.

The user exit NOCPRINT can be written in any language which supports the register conventions described above. It must be linked to the Natural nucleus together with the Natural Optimizer Compiler nucleus.

Working with the PGEN Output

This section provides hints and explanations on how to interpret the output created with the PGEN option.

- At the top of the PGEN output are some disassembled lines which do not appear to belong to any source line. These are the instructions which make up the prologue, which is executed whenever control passes from non-optimized to optimized code. Permanent base registers are loaded and control is passed to the correct point in the prologue. See *Example Section A* below.
- Sometimes a lot of source lines are printed without any code. This indicates that there was no code required or that these statements are excluded from the NOC optimization. See *Example Section B* below.

  Moreover, when the code generated for a Natural statement consists only of:

```
BAS   R14,RETH
DC    X'....'
```

  this indicates a return back to the standard runtime, as this statement could not be NOC optimized (see line 0170).

- If the NODBG=OFF (default) has been specified, a sequence of instructions is generated at the start of each Natural statement:

```
BALR R9,R11
DC X'....'
```

  This sequence sets the line number (in case of error) and checks whether the TEST mode is switched ON. Without this sequence, debugging of NOC-compiled statements by the Natural Debugger is not possible. See *Example Section C* below.

- Sometimes there is a line break between disassembled lines. This break indicates an internal statement separation. It happens because often a single Natural statement will generate multiple internal (pseudo-code) statements.
- The Natural variables operated are inserted in the Assembler code.
- The items on the right side (e.g. "START 8FEC") are of internal nature. They document the path how the code was generated by the NOC modules.
- All kind of addresses inside the code are resolved and provided in the form "`=(00044)`". It documents the offset in the code to which the branch is executed.
- The first and the last code instruction contains the NOC version used to compile this program. The meaning of "`4700 8410`" is NOC V841.

Example Section A:

```
000000 4700 8410           NOP   1040(,R8)                          START  8FEC
000004 5880 D354           L     R8,CONST                                 D9DC
000008 5870 D370           L     R7,LOCAL                                 D9DC
00000C 4810 6006           LH    R1,6(,R6)                                90A0
000010 1F60                SLR   R6,R0                                    90BA
000012 47F1 A000           B     0(R1,R10)                                90C0

000016 4DE0 B040           BAS   R14,RETH                          RETN   F0AA
00001A 0034                DC    X'0034'                                  F0C0
```

Example Section B:

```
0010 0010 OPTIONS MCG=(PGEN,OVFLW,INDX)
 0020 DEFINE DATA LOCAL
 0030 1 I(I4)
 0040 1 P(P7.2)
 0050 1 T(P7.2)
 0060 END-DEFINE
 0070 *
 0080 SETTIME
 0090 *
```

Example Section C:

```
0100 FOR I=1 TO 100000                                                  ↩


00001C 0D9B                BASR  R9,R11                           MOVE 1724A
00001E 004A                DC    X'004A'                               17278
000020 D203 7000 8148      MVC   I(4),#KST0148                          97D2

000026 47F0 A044           B     68(,R10)         =(00044)          GOTO  EF44

00002A 0D9B                BASR  R9,R11                            ADD 1724A
```

```
00002C 006A               DC    X'006A'                                 17278
00002E BF0F 7000          ICM   R0,B'1111',I                              BB20
000032 5A00 8148          A     R0,#KST0148                               1E4E
000036 0D90               BASR  R9,0                                      F12A
000038 4710 B15C          BO    NAT1301                                   1E9E
00003C BE0F 7000          STCM  R0,B'1111',I                              A9CC

000040 0D9B               BASR  R9,R11                              IF    1724A
000042 007C               DC    X'007C'                                   17278
000044 BF0F 7000          ICM   R0,B'1111',I                              BB20
000048 5900 819B          C     R0,#KST019B                               3FDA
00004C 47D0 A054          BNH   84(,R10)          =(00054)                EF44

000050 47F0 A078          B     120(,R10)         =(00078)          GOTO  EF44   ↩


0110   ADD 1.00 TO P

000054 0D9B               BASR  R9,R11                              ADD   1724A
000056 0092               DC    X'0092'                                   17278
000058 FA41 7004 819F     AP    P(5),#KST019F(2)                          20A0
00005E 0D90               BASR  R9,0                                      F12A
000060 4710 B15C          BO    NAT1301                                   1071C
000064 910D 7008          TM    P+4,X'0D'                                 120B0
000068 4710 A070          BO    112(,R10)         =(00070)                120F6
00006C 960F 7008          OI    P+4,X'0F'                                 1210ª

0120 END-FOR
0130 *

000070 0D9B               BASR  R9,R11                              GOTO  1724A
000072 00A4               DC    X'00A4'                                   17278
000074 47F0 A02A          B     42(,R10)          =(0002A)                EF44   ↩


0140 T:=*TIMD(0080)

000078 0D9B               BASR  R9,R11                              SYFU  1724A
00007A 00AE               DC    X'00AE'                                   17278
00007C 4DE0 B0D8          BAS   R14,SYSFUNC                               5F1A
000080 0190 B881          DC    X'0190B881'                               5F28

000084 F246 7009 8190     PACK  T(5),#KST0190(7)                    MOVE  AD18
00008A 910F 700D          TM    T+4,X'0F'                                 12130
00008E 4710 A0A0          BO    160(,R10)         =(000A0)                12176
000092 17EE               XR    R14,R14                                   1218E
000094 43E0 700D          IC    R14,T+4                                   12194
000098 43EE B488          IC    R14,PSGNTR(R14)                           121AA
00009C 42E0 700D          STC   R14,T+4                                   121B2
0000A0 F040 7009 0002     SRP   T(5),2,0                                  ACA2
0000A6 17EE               XR    R14,R14                                   1218E
0000A8 43E0 700D          IC    R14,T+4                                   12194
```

```
0000AC 43EE B488           IC    R14,PSGNTR(R14)                          121AA
0000B0 42E0 700D          STC    R14,T+4                                 121B2   ↩


0150 T:=T / 10
0160 *

0000B4 0D9B               BASR   R9,R11                               DIV 1724A
0000B6 00C0               DC     X'00C0'                                 17278
0000B8 F864 D100 7009     ZAP    OP1(7),T(5)                              AC60
0000BE FD61 D100 81A1     DP     OP1(7),#KST01A1(2)                       327A
0000C4 F844 7009 D100     ZAP    T(5),OP1(5)                              AC60
0000CA 910D 700D          TM     T+4,X'0D'                               120B0
0000CE 4710 A0D6          BO     214(,R10)        =(000D6)               120F6
0000D2 960F 700D          OI     T+4,X'0F'                              1210A   ↩


0170 DISPLAY 'ELAPSED TIME (S)' T

0000D6 4DE0 B040          BAS    R14,RETH                          RETN  F0AA
0000DA 00D2               DC     X'00D2'                                 F0C0

0180 END

0000DC 40D6 D7E3 F844 2000 DC    X'40D6D7E3F8442000'  =' OPT8à..'    END  927E
0000E4 0000 0000          DC     X'00000000'                               nf
0000E8 40D5 D6C3 F8F4 F140 DC    X'40D5D6C3F8F4F140'  =' NOC841 '         92E4
```

Influence of other Natural Parameters

The global parameter ZD influences the behavior of the NOC compiler. See the description of the ZD option as described under ***List of Options*** above.

The COMPOPT parameter PSIGNF (see also the system command COMPOPT in the Natural *System Commands* documentation) influences the behavior by forcing the signs of positive packed decimal numbers to F if ON, and to C if OFF. The parameter is applied if NOSGNTR=OFF is specified.

See the chart below for packed data (Format P) ":"

NOSGNTR=OFF	and	PSIGNF=ON	All signs are normalized to F (default).
NOSGNTR=OFF	and	PSIGNF=OFF	All signs are normalized to C.
NOSGNTR=ON			All signs are left as they were generated by the last operation.

For numeric data (Format N) the signs are always normalized to F, regardless of the settings of NOSGNTR and PSIGNF.

9 Performance Considerations

Formats

Best performance is achieved when you use the data formats packed numeric (P) and integer (I4) in arithmetic operations.

Avoid converting data between the formats packed numeric (P), unpacked numeric (N), integer (I), and floating point (F), as this causes processing overhead even with optimized code.

As there is no interpretation overhead with optimized code, the differences between the various data formats become much more prominent: with optimized code the performance improvement gained by using format P instead of N, for example, is even higher than with normal code.

Example:

```
A = A + 1
```

In the above numeric calculation

- with non-optimized code, format P executes approximately 13 % faster than format N.
- with optimized code, however, format P executes approximately 56 % faster than format N.

The performance gain which would be achieved by applying the Natural Optimizer Compiler to this simple statement is

- with unpacked operands (N): 8 times faster
- with packed operands (P): 15 times faster

Arrays

Array range operations, such as

```
MOVE A(*) TO B(*)
```

are executed more efficiently than if the same function were programmed using a `FOR` statement processing loop. This is also true for optimized code.

When indexes are used, integer format I4 should be used to achieve optimum performance.

Alphanumeric Fields

We recommend that you adjust the length of the alphanumeric constant to the length of the variable, when moving an alphanumeric constant to an alphanumeric variable (format A), or when comparing an alphanumeric variable with an alphanumeric constant. This will significantly speed up operation, for example:

```
A(A5):='XYZAB'


...
IF A = 'ABC  ' THEN ...
```

is faster than

```
IF A = 'ABC' THEN ...
```

DECIDE ON

When using the `DECIDE ON` statement with a system variable, array or parameter *operand1*, it is more efficient to move the value to a scalar variable of the same type and length defined in the `LOCAL` storage section.

Numeric Values

When using numeric constants in assignments or arithmetic operations, try to force the constants to have the same type as the operation.

Rules of Thumb

- Any numeric constant with or without a decimal but without an exponent is compiled to a packed number having the minimum length and precision to represent the value, unless the constant is an array index or substring starting position or length, in which case it becomes a four-byte integer (I4). This rule applies irrespective of the variable types participating in the operation.
- Operations containing floating point will be executed in floating point. Add `E00` to numeric values to force them to be floating point, for example:

```
ADD 1E00 to F(F8)
```

- Operations not containing floating point, but containing packed numeric, unpacked numeric, date or time variables will be executed in packed decimal. For `ADD`, `SUBTRACT` and `IF`, force numeric constants to have the same number of decimal places as the variable with the highest precision by adding a decimal place and trailing zeros, for example:

```
ADD 1.00 TO P(P7.2)
```

This technique is unnecessary for `MULTIPLY` and `DIVIDE`.

Variable Positioning

To ease the optimization process, try to keep all scalar references at the front of the data section and all array references at the end of the data section.

Variable Caching

The Natural Optimizer Compiler contains an algorithm to enhance the performance even further. In terms of performance, a statement will differ depending on the types of operands. The statement will execute more slowly if one or more of the operands is a parameter, array or scalar field of Type N (numeric) or combinations of these operands. The NOC analyzes the program flow and determines which variables with one or more of these characteristics are read two or more times without being written to. It then moves the value of each variable to a temporary cache area where it can be accessed quickly under the following conditions:

- The variable is accessed often but seldom modified *and*
- The variable is an array of any type or a scalar field of Type N (numeric).

Most suitable for variable caching are programs with long sequences that repeatedly access the same variable, in particular if the variable is an array. Variable caching then avoids complex and recurring address computation.

Example of Variable Caching

The example program displayed below demonstrates the advantage of variable caching. Cataloged with NODBG (see below) and CACHE=ON, executing this program in a test environment took 47 % of the time required to execute the program with NODBG and CACHE=OFF. Cataloging the program with CACHE=ON, reduces the code generated by the NOC from 856 bytes to 376 bytes.

```
DEFINE DATA LOCAL
1 ARR(N2/10,10,10)
1 I(I4) INIT <5>
1 J(I4) INIT <6>
1 K(I4) INIT <7>
END-DEFINE
DECIDE ON EVERY ARR(I,J,K)
 VALUE 10 IGNORE
 VALUE 20 IGNORE
 VALUE 30 IGNORE
 VALUE 40 IGNORE
 VALUE 50 IGNORE
 VALUE 60 IGNORE
 VALUE 70 IGNORE
 VALUE 80 IGNORE
 VALUE 90 IGNORE
 NONE IGNORE
END-DECIDE
```

Caution: If the content of a cached variable is modified with the command MODIFY VARIABLE of the Natural Debugger, only the content of the original variable is modified. The cached value (which may still be used in subsequent statements) remains unchanged. Therefore, variable caching should be used with great care if the Natural Debugger is used. See also the Natural *Debugger* documentation.

NODBG

Once a program has been thoroughly tested and put into production, you should catalog the program with the NODBG option as described in the section ***Optimizer Options***. Without debug code, the optimized statements will execute from 10% to 30% faster.

The code to facilitate debugging is removed when this option is specified, even with INDX or OVFLW options turned on.

10 Listing Zaps

If you want to have an overview of the Zaps that have been applied to the Natural Optimizer Compiler at your site, use the `DUMP` system command.

To obtain a Zap overview

- Enter the Natural system command

  ```
  DUMP ZAPS NOC
  ```

  A list of the Zaps that have been applied is displayed.

If no Zaps have been applied to the Natural Optimizer Compiler, you will receive the appropriate message.

IV Natural Optimizer Compiler Version 8.3/8.4 - Documentation Updates

11 Natural Optimizer Compiler Version 8.3/8.4 - Documentation Updates

Note: The documentation updates provided here only cover the changes introduced in Natural Optimizer Compiler Version 8.3 and Version 8.4.

For the changes in installation, see *Installing the Natural Optimizer Compiler* on z/OS, z/VSE and BS2000 in the Natural *Installation* documentation.

Optimizer Options under Natural Optimizer Compiler Version 8.3/8.4

Note: This is an extract of the chapter ***Optimizer Options*** and only describes the changes specific to the Natural Optimizer Compiler Version 8.3 and Version 8.4.

ARCH Option

The ARCH option specifies the hardware architecture level to be used for generating code for executable Natural objects.

When you specify an ARCH value, the Natural Optimizer Compiler generates newer and faster machine instructions that can improve the performance of the generated code. You cannot specify a value that is higher than the architecture level of your current machine. An executable Natural object cataloged with an ARCH level can only run on a machine with the same or a higher architecture level. Therefore, we recommend not to use the ARCH option if the cataloged objects are intended to execute on any machine, especially on a machine with a lower architecture level (for example, BS2000).

For detailed information on architecture levels, see the related literature from IBM (z/Architecture, Principles of Operation).

The following architecture levels are supported by the ARCH option of the Natural Optimizer Compiler:

Architecture Level	IBM Hardware Facility Required
0	Specifies that no architecture level is used. This is the default setting for compatibility with all mainframe platforms supported by Natural.
1 to 4	These values are not evaluated and treated as ARCH=0.
5 to 6	■ z800 or z900 Extended-Translation Facility 2 ■ z890 or z990 HFP Multiply-and-Add/Subtract Facility

Architecture Level	IBM Hardware Facility Required
7	■ z9 to z109 Extended-Immediate Facility
8	■ z10 General-Instructions-Extension Facility Execute-Extensions Facility
9	■ zEnterprise 196 Load/Store-on Condition Facility Floating-Point-Extension-Facility Distinct-Operands Facility High-Word-Facility
10	■ zEnterprise EC12 (zEC12) Decimal Floating-Point Facility Decimal Floating-Point Zoned-Conversion Facility
11	■ zEnterprise z13 Decimal Floating-Point Packed-Conversion Facility
12	(Applies to Natural Optimizer Compiler Version 8.4 and above only.) ■ zEnterprise z14 Vector Packed-Decimal Facility

Note: With an `ARCH` value greater zero, the Natural Optimizer Compiler generates instructions up to the facility level described in the table above. An `ARCH` value higher than the architecture level of the underlying machine is rejected at compile time. The attempt to start a program compiled with an `ARCH` level on a machine with a lower architecture level, causes a NAT1394 runtime error. You can display information on the current machine by using the `TECH` system command.

This section covers the following topics:

- Support for Architecture Level 10
- Support for Architecture Level 11
- Support for Architecture Level 12
- Compatibility for Architecture Level 10 and 11

Support for Architecture Level 10

When ARCH=10 is set, the Natural Optimizer Compiler generates instructions provided by the Decimal-Floating-Point (DFP) Zoned-Conversion Facility for the numeric operations described in the following section. This can significantly improve the execution speed for statements that use these operations.

Operations Optimized by ARCH=10

The following arithmetic operations on variables of the Natural data formats I (integer), N (numeric unpacked) and P (packed numeric) benefit from ARCH=10:

- Value assignments:

  P:=I

  P:=N

  N:=I

  N:=N

  N:=P only if the number of packed digits is less than or equal to 15.

  I:=N

- Arithmetic operations, such as ADD, SUBTRACT, DIVIDE and MULTIPLY statements, but only if both of the following conditions apply:

  At least one of the operands used is in the format N or I.
  The operation result does not exceed 34 (integer + precision) digits.

- Comparisons, such as IF and DECIDE statements, but only if both of the following conditions apply:

  At least one of the operands used is in the format N.
  Both operands are in different formats.

Support for Architecture Level 11

When ARCH=11 is set, the Natural Optimizer Compiler uses machine instructions introduced with the DFP Packed-Conversion Facility. In addition to the numeric operations optimized with ARCH=10, ARCH=11 also optimizes operations that use packed variables only.

Support for Architecture Level 12

Applies to Natural Optimizer Compiler Version 8.4 and above only.

When `ARCH=12` is set, the Natural Optimizer Compiler generates machine instructions introduced with the Vector Packed-Decimal Facility (VPD) in the z14 hardware class. This can improve the execution speed for assignments, comparisons, and calculations if at least one packed operand is used.

VPD machine instructions are generated for the same Natural operations described for **Architecture Level 11**, except they are applied only to arithmetic operations whose results do not exceed 31 (integer + precision) digits.

Compatibility for Architecture Level 10 and 11

When `ARCH=10` is used, the Natural Optimizer Compiler generates machine instructions introduced with the Decimal-Floating-Point (DFP) Zoned-Conversion Facility or the DFP Packed-Conversion Facility. These instructions execute faster than the standard machine code instructions for arithmetic operations, but they do not accept data which is improper in terms of the zoned numeric data type (N).

This may cause runtime errors, when an N-field is defined within a `REDEFINE` section of an alpha or binary variable and the N-field is not properly initialized before used in an arithmetic operation.

A numeric zoned field carries one digit in one byte. Usually, each byte contains x'F' in the left halfbyte (**Z**one bits) and the digit value (0-9) in the right halfbyte (**N**umeric bits). This applies for all bytes, except for the last one, which contains (A-F) in the left halfbyte (**S**ign bits).

A sign halfbyte (C,A,F,E) represents a positive value, whereas (B,D) stands for a negative value. A value other than (0-9) inside the numeric halfbytes (N) and a value other than (A-F) inside the sign halfbyte (S) is considered invalid. The data inside the zone halfbytes (Z) is not regarded by arithmetic conversion instructions and can have any value (0-F).

Example for a variable defined as (N6):

ZN	ZN	ZN	ZN	ZN	SN	Sign is	Value is	Works with `ARCH<=9`	Works with `ARCH>=10`
F1	F2	F3	F4	F5	F6	F=positive	123456, ok	yes	yes
F3	F2	F6	F3	F3	D2	D=negative	323662, ok	yes	yes
40	40	40	40	40	40	4=invalid	000000, ok	yes	NAT7024
00	00	00	00	00	00	0=invalid	000000, ok	yes	NAT7024
12	13	14	15	16	17	1=invalid	234567, ok	yes	NAT7024
51	6B	72	7A	12	F1	F=positive	1B2A21, invalid	NAT0954	NAT7024

When ARCH=9 (or lower) is used, invalid sign halfbytes (0-9) are automatically corrected by the generated code to a positive sign (F). This turns N-fields with a blank contents into valid data with value zero. The same applies for Hexa zero data.

When ARCH=10 (or 11) is used, invalid sign halfbytes (0-9) remain unchanged and lead to a program check (Data exception) when accessed by a DFP instruction. If such an abend occurs, Natural issues a NAT7024 error instead of a NAT0954 to clearly indicate that the error is caused by an N-variable that does not contain valid numeric data.

If a numeric halfbyte (N) contains a value other than (0-9), a program check (Data exception) happens regardless of the ARCH level used.

Conclusion:

Do not use ARCH=10 (or 11) to catalog a program which operates unclean numeric data, with a sign value other than (A-F).

For example:

```
OPTIONS MCG=(PGEN,ARCH=9)
DEFINE DATA LOCAL
1 #A (A6)
1 REDEFINE #A
  2 #N (N6)
END-DEFINE                                    /*  ARCH=9  ARCH=10
#A := H'F1F2F3F4F5F6'   ADD 1 TO #N  WRITE #N  /*  ok       ok
#A := H'F3F2F6F3F3D2'   ADD 1 TO #N  WRITE #N  /*  ok       ok
#A := H'404040404040'   ADD 1 TO #N  WRITE #N  /*  ok       NAT7024
#A := H'000000000000'   ADD 1 TO #N  WRITE #N  /*  ok       NAT7024
#A := H'121314151617'   ADD 1 TO #N  WRITE #N  /*  ok       NAT7024
#A := H'516B727A12F1'   ADD 1 TO #N  WRITE #N  /*  NAT0954  NAT7024
END
```

Moreover, when ARCH=10 (or 11) is used, Natural can issue a NAT1305 (truncated numeric value) instead of a NAT1301 error (intermediate result too large) for the following reason: The DFP numeric format is used for calculating intermediate results and an overflow is only detected at the end of the arithmetic operation when the DFP is converted into the format of the result.

Compatibility for Architecture Level 12

When ARCH=12 is used, the Natural Optimizer Compiler generates machine instructions introduced with the Vector Packed-Decimal Facility (VPD) which are compatible in terms of data incorrectness with the code generated with ARCH=9 or below.

Numeric data fields (N) with incorrect sign representations (0-9) are converted into the positive sign value (F). This accepts numeric fields with a blank or hex00 content and treats them as value zero. A data exception (abend) does not occur in these cases.

Prerequisites for Code Generation with Unicode Operands

The Natural Optimizer Compiler generates optimized code for Natural statements with Unicode strings if the following requirements are met:

Statement	Requirement
All statements	All operands used in the statement must be of the type Unicode.
EXAMINE	The ARCH option must be set to a value greater than or equal to 6.
IF DECIDE FOR DECIDE ON	■ All Unicode character strings must be normalized. ■ The ARCH option must be set to a value greater than or equal to 5. ■ The UNICC **option** must be set to ON or FORCE. ■ The COLLATE option of the CFICU profile parameter must be set to OFF (see the *Parameter Reference* documentation).
MOVE MOVE SUBSTRING RESET	The ARCH option must be set to a value greater than or equal to 5.

UNICC Option

The UNICC option controls the generation of optimized code for IF, DECIDE FOR and DECIDE ON statements that contain Unicode operands.

Valid values for UNICC are:

Value	Explanation
ON	Generates optimized code and checks whether COLLATE=OFF is set (see the CFICU profile parameter in the *Parameter Reference* documentation). If COLLATE=ON is set, execution of the optimized code will fail with a NAT7023 Natural system error.
FORCE	Generates optimized code analogous to ON but without COLLATE=OFF check. The code optimized with FORCE performs better than the code optimized with ON but can cause wrong results if COLLATE=ON is set.
OFF	Optimized code is not generated. OFF is the default setting.